In *Proc. 1996 IEEE Intl. Conf. on Robotics and Automation*, pp. 3361–8

The Archimedes 2 Mechanical Assembly Planning System [*]

Stephen G. Kaufman Randall H. Wilson Rondall E. Jones Terri L. Calton
Arlo L. Ames

Intelligent Systems and Robotics Center
Sandia National Laboratories
Albuquerque, NM 87185-0951

Abstract

We describe the implementation and performance of Archimedes 2, an integrated mechanical assembly planning system. Archimedes 2 includes two planners, two assembly sequence animation facilities, and an associated robotic workcell. Both planners use fully 3 dimensional data. A rudimentary translator from high level assembly plans to control code for the robotic workcell has also been implemented. We can translate data from a commercial CAD system into input data for the system, which has allowed us to plan assembly sequences for many industrial assemblies. Archimedes 2 has been used to plan sequences for assemblies consisting of 5 to 109 parts. We have also successfully taken a CAD model of an assembly, produced an optimized assembly sequence for it, and translated the plan into robot code, which successfully assembles the device specified in the model.

1 Introduction

The *mechanical assembly planning* problem is that of taking a model of a mechanical device, and computing one or more assembly sequences that will assemble the device from its piece parts. It is actually a wide class of problems, depending on the constraints that the resulting sequence must satisfy. The minimum constraint is that the sequence must be geometrically valid — that is, none of the operations of joining parts and subassemblies require interpenetration of parts. Another constraint is that any tools that are required to perform assembly operations must be able to go through a sufficient range of motion to fulfill their function. We may also require that the available assembly operations be limited to those that can be performed by a given assembly apparatus. This makes possible an automatic assessment of the assemblability of a product on an existing assembly system.

Because of the importance of the mechanical assembly planning problem, the last decade has witnessed an explosion in computer-aided assembly planning research (see, e.g., [2]). Significant advances have been made in both theory and practice, and a great number of experimental systems have been built.

Successful solution of the problem would be of great use to product and manufacturing system designers. A system implementing such a solution would enable design for assembly, by providing analysis of the assemblability of a design as expressed by a CAD model. It would therefore have the potential to significantly reduce the costs of product realization. Output from such a system could be of several forms; instructions to a human assembler, an animation of the assembly sequence, or an animation of an assembly system carrying out the assembly. The utility of such a system would be even greater if the resulting plans could be translated directly into control software for an assembly apparatus, such as a robotic workcell.

This paper describes a system we have implemented, called Archimedes 2, that is a first step towards such an integrated mechanical assembly planning system (the system is a successor to the prototype Archimedes system developed at Sandia [13]). Its two planners have computed sequences for assemblies from both industry and government; one planner has done so for an assembly of 109 parts. We have also demonstrated a complete and automatic path from a

[*]This work was supported by Sandia National Laboratories under DOE contract DE-AC04-94AL85000, and by the Laboratory Directed Research and Development Office of Sandia National Laboratories.

CAD model of a mechanical assembly, to an optimized assembly sequence, to an automatically programmed robot that carries out the sequence. To our knowledge, no other research program has demonstrated this completeness. Two assembly sequence animation facilities have been developed, one for showing a sequence as "flying parts", the other for showing a simulated workcell performing the sequence. These animations are driven directly by the output of the assembly planners.

The rest of this paper, a substantially abbreviated version of [6], is organized as follows. We first describe the system, both its overall architecture and the implementation of its components. Then we describe the results of running it on various assemblies. Finally, we describe our current and future work on the system. Related work is referenced throughout the paper where appropriate.

2 System Description

Archimedes 2 includes two assembly planners. One considers part geometry and tool accessibility, and finds a single plan if one exists. The actual planning architecture is very simple, and its power derives almost exclusively from the large suite of geometric functions and queries we have developed. Its success demonstrates the power of this suite of geometric functions, and we therefore call this planner the "geometric engine".

The second planner is built on top of the geometric engine, using a standard search algorithm that simultaneously considers many assembly sequences. It also considers the orientation of the current subassembly and the grippers needed to acquire the parts, preferring sequences that minimize inversion operations and gripper changes. Currently, it does not consider tool accessibility.

Both of these planners require geometric models of the complete assembly. Additional data about part joinings (e.g., welds and pressfits) is also necessary.

Besides the two planners, the Archimedes 2 software includes an animation facility ("Illustrator"), a translator of high-level plans into robot control code, and a library of robot control subroutines. The system also includes an Adept 2 robot as the target of this translation. Related to Archimedes 2, though not part of the core system, is a "design module" that provides CAD data. This module is based on the Pro/ENGINEER® CAD system. Because of space limitations, we will not describe it in this paper.

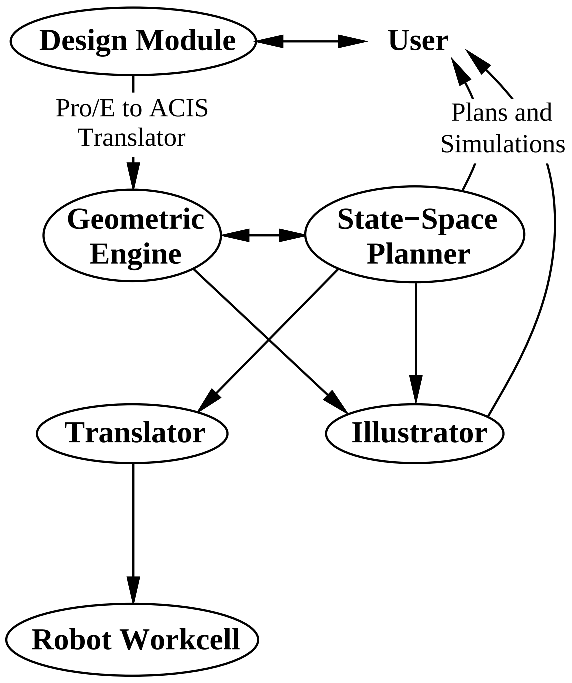

Figure 1: The architecture of Archimedes 2

The relationships between the components are illustrated in figure 1.

We will first describe the assumptions we made in developing the software, and then the inputs and outputs to the Archimedes 2 system. Then, each system component will be described in greater detail.

2.1 Background and Assumptions

Our planners produce two-handed, monotone assembly sequences. The sequences do not need to be linear: subassemblies can be identified as part of the input, or be computed.

Our software uses the ACIS® solid modeling system to represent the geometry and topology of parts and assemblies. ACIS uses a boundary representation for solids; it has a facetting capability, which we use in an essential way. Each body is facetted as part of model input; this facetted representation is used for contact-finding, collision detection, and animation.

Archimedes 2 encodes the following three major assumptions:

- All parts are perfectly rigid.
- The available assembly operations are

  mating, which puts a part in its correct relative position (with respect to the other parts in the assembly). This operation requires specification of the pair of parts or subassemblies to be mated, and a trajectory for one of them to follow.

joining, which attaches two parts once they are in the correct relative position. These include welding, screwing, glueing, etc.

inverting a part or a subassembly.

- Mating trajectories are translations, rotations, or a combination (as for driving a screw).

2.2 System Input

Input to the Archimedes planners is of two types: 1) solid models of parts (in their assembled configurations); and 2) additional information about how parts are joined, recommended subassemblies, and suggested assembly directions. These types of inputs are gathered together into an *assembly file.* An assembly file contains a list of filenames describing parts or subassemblies; associated with each is a 4×4 transformation matrix that specifies the location of the part or assembly in the world coordinate system. Files describing parts are ACIS-readable models. Files describing subassemblies have the same form as the top-level assembly file.

Each assembly or subassembly has an associated file describing non-geometric data. Such non-geometric data is a standard feature of assembly planning systems. Bourjault's *liaisons* [1] and Ko and Lee's *mating conditions* [7] are early examples, while Homem de Mello's *relational model* [3] is probably the most comprehensive view. Work on CAD standards such as PDES/STEP [4, 9] promises to standardize representations for much of this information.

Specifically, the auxiliary files declare:

- threaded contacts, pressfits, and snapfits. The geometric routines will correctly report part overlap for such matings, leading to the inference that the matings cannot be realized. Yet the matings are indeed realizable, via screwing motions and part deformation, respectively. The planners simply assume that any such matings will be made successfully, and perform no geometric checking for them. The geometric engine requires that directions be specified for such matings.
- Recommended subassemblies.
- Which tools are needed to perform certain assembly operations, so that their accessibility can be checked. Due to lack of space, we cannot describe that work in the present paper; see [15] for a description.

The reader may wonder why some of the part-part contacts must be identified by the user as input to the system. The answer has to do with finding the proper balance between effort on the part of the human and the machine. It is very easy for the human to recognize and specify these contacts, but very difficult to develop a feature-recognition algorithm that will do the same. CAD systems of the future will probably allow such specification of contact types. Our experience with data from Pro/ENGINEER, in which such contact information is not available, is that it takes hours (not tens of hours) to find and specify such contacts. Compare this effort with the months that would be required to program the appropriate feature recognition functions, which would still fail in some cases.

2.3 System Output

The Archimedes 2 system has several forms of output. From the least to the greatest level of detail, they are:

1. Textual plans, using the three operations above.
2. Animations of plans, showing parts being brought together without any tooling or fixturing (this is like an animated exploded diagram). These animations can also include inversion operations.
3. Animations of workcells performing the assembly, with robots, tooling, and fixturing.
4. Robot control code sufficient for robotic assembly of the given device.

Animations and robot code outputs will be described below. The textual plans include the following parameters: for mating operations, the target (fixture, or another part) and the gripper; for inversion operations, the subassembly to be inverted; for welding, the position of the weld (we assume laser spot welds).

2.4 System Components

2.4.1 Geometric Engine

The geometric engine is an implementation of the methods of [14] in C++ using ACIS. It determines sequences of assembly operations that are geometrically valid, i.e., those in which parts do not collide with each other.

Contact Finding Before planning begins, all part-part contacts are found by computing all face-face contacts. Input models include planar, spherical, cylindrical, conical, and toroidal faces, leading to 25 possible face contact types. Of these, we have implemented only four: plane-plane, plane-cylinder, plane-sphere, and cylinder-cylinder. In addition, a *threaded* contact type is defined, which is used to model the contact between a screw and the threads into which it is turned. These five contact types have been sufficient to capture virtually all of the contacts found in the examples described below. The few unsupported types we have encountered have not affected the geometric correctness of the assembly sequences computed.

Each contact type has a specialized routine to determine if a contact of that type exists between two faces. The facetted representation is frequently used in these routines.

Trajectory Selection Part matings are used to select possible mating trajectories. For example, if two planar faces of distinct parts are in contact, the feasible infinitesimal trajectories of each part are restricted to those such that the scalar product of the trajectory vector and the outward-pointing contact normal of the moving part is non-positive. The allowable trajectories of a part, given the totality of its contacts with other parts, is the intersection of the sets of trajectories allowed by each contact. This information is stored in a structure called a *non-directional blocking graph*, or *NDBG* [16], which is used to compute feasible partitionings of the assembly into subassemblies, and to select trajectories for parts or subassemblies.

Trajectory Checking If a part can move at all, the distance it can move before colliding with any other part must be sufficient to remove it completely from the rest of the assembly. An obvious way to do this is with volume sweeping computations, such as those provided by ACIS. When parts have complex geometries, however, such methods are prohibitively slow. We therefore developed a new method of collision detection, which exploits the hardware capabilities of a graphics workstation. The method is raster-based, discretizing the silhouettes of the (facetted representation of) parts in a plane perpendicular to the direction of motion, but otherwise makes no simplifying assumptions or approximations.

A brief summary of the available graphics hardware is required to understand how the method works. The displayed image has color intensity values stored in a hardware buffer, called the image buffer, for each pixel. Another hardware buffer, called the z-buffer, stores the *distance* of each pixel from the viewpoint. This buffer is typically used in rendering 3D scenes in which occluded objects should not be displayed; it works as follows. Each pixel of the z-buffer is initialized to the maximum distance. For each object displayed, each pixel that will be in its displayed image is checked for its distance to the viewpoint. If it is less than the current distance stored in the z-buffer, the image buffer is updated with the intensity value of the pixel. If it is greater than the current distance, the object is occluded, and so the image buffer is not updated.

The graphics hardware allows the distance comparison to be a greater-than test instead of less-than. This setting can be modified from the program; it is the key to the fast checking. To test if part A will collide with part B when A is translated to infinity, the z-buffer is first cleared. The viewpoint is set to be the opposite of the sweep direction, so the sweep is effectively away from the viewpoint. Part A is drawn in black, then part B is drawn in white, with the greater-than distance comparison. Any pixels of part B that are outside the boundary of part A will not be drawn, since they are not further than the maximum distance to which the z-buffer has been initialized. If any pixels of part B are drawn, it is because they are further than the pixels of part A; since we are viewing along the sweep direction, this means that part A will collide with part B at some point along an infinite translation. Therefore, we need only check for the existence of white pixels in the image buffer. This is easily done using the programming interface to the graphics system. Figure 2 illustrates the technique.

These operations are all performed in hardware. Translations of any 3D direction, and 3D parts of any shape can be tested, since the graphics hardware applies the proper transformations during rendering. We have observed that the planner runs nearly three orders of magnitude faster when using the hardware-based method than when using the volume sweeping method.

Planning with the Geometric Engine The geometric engine has a very simple planning strategy. Parts or subassemblies are tested for removability in turn; if one may be feasibly removed, the planner recurses on the remaining subassembly. When all parts have been removed, the recursion is unwound, resulting in a sequence of pairs of parts or subassemblies and trajectories.

2.4.2 Optimizing Planner

The optimizing planner considers many possible disassembly sequences, and finds a sequence of lowest cost according to a given cost function. The planner

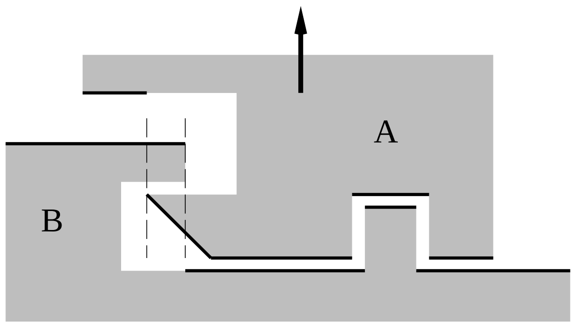

Figure 2: A 2D cross-section of the z-buffer collision detection test for translating 3D parts. Part A is to be translated in the direction shown. Bold lines indicate part silhouettes. Because B's silhouette is above A's between the dotted lines, a collision will occur.

is based on an algorithm that searches a tree structure of partial assembly decompositions of the given assembly. Each path through the tree represents a different partial assembly sequence; at each search step, the current best partial sequence is decomposed further. This process repeats until the decomposition is complete. The search algorithm used is an implementation of the A* state-space search algorithm [12], which is guaranteed to find a path of minimum cost under certain circumstances.

Assumptions The optimizing planner makes the following assumptions, which are consistent with the target workcell to which we translate high level plans.

1. Subassemblies may be in one of two orientations, normal and inverted. One orientation is obtained from the other by rotating the subassembly 180 degrees. We presume that there is a device in the workcell that can perform this inversion operation.
2. All insertion operations are translations in the downward direction.
3. There are sufficient grippers in the workcell to manipulate each part; the mapping between parts and grippers is given to the planner.
4. All joining operations can be performed by a laser spot welder. Although our workcell does not have such a welder, it performs assembly as though it did; welding operations are simulated by replacing unwelded assemblies with welded assemblies.

States A *state* is a data structure representing: 1) the parts that have not yet been removed during the planning process; 2) the current orientation of the subassembly consisting of these unremoved parts; and 3) the number of gripper changes and subassembly inversions needed to assemble the remaining parts (these two values are functions of the path to the state, not the state itself).

Search Strategy Search begins with the complete assembly. For each part not in a recommended subassembly, and each recommended subassembly, a state is generated corresponding to the removal of the part or subassembly. These states are tested for geometric feasibility, and for visibility of all weld sites to the laser welder (geometric feasibility testing is implemented by the geometric engine). If a state fails these tests, it is inverted, and the tests are reapplied. This is the only way inversions are introduced into the plan. If the tests fail again, the state is deleted.

In either case, if the tests succeed, the number of gripper changes and the number of inversions are updated, by comparing the gripper needed for the removal to the previously used gripper, and comparing the orientation of the successor state to that of the given state. If either comparison fails, the respective count is incremented by one. The cost of the state is then computed as the sum of these two values, and the number of remaining parts. This cycle repeats with the lowest cost state. Search terminates when a state with no unremoved parts is found, or when there are no states left to expand. The first case corresponds to success, the second to failure. In case of success, the list of states is transformed into the corresponding list of actions.

When the plan for the assembly is complete, a plan is computed for each recommended subassembly. This continues recursively until all subassemblies have been planned, or one is encountered for which a plan cannot be found. Note that when recommended subassemblies are used, the search becomes a hybrid approach, not a pure A* search. That is, the search is not over as large a space as would be required to assure optimality. The tradeoff between computational time and optimality seems unavoidable. In practical terms, guiding the planner by suggesting reasonable subassemblies seems workable.

2.4.3 Code Generator

The code generator (or translator) produces a sequence of function calls in Adept's robot programming language, V+, that implement the given plan in our workcell. Inputs to the translator are the high level plan, a file describing the location of objects (such as fixtures) in the workcell, and a file describing grasp points of the parts.

The translation is straightforward, essentially a process of replacing high-level operations with calls to their counterparts in the V+ library, described below. Parameters of these functions are computed, or read from files. This use of skeletal procedures has been used in LAMA [10] and AUTOPASS [8] (unlike those systems, we do not simulate the proposed operations to assess their likelihood of success).

2.4.4 Workcell

The target of the translation process is a workcell consisting of an Adept2 robot, two arm-mounted cameras, a force sensor, a quick-change wrist, a parallel jaw gripper with built-in RCC, a vacuum gripper, an inverter, a parts kit, and three fixtures. The vision system and force sensor controller are both Adept products.

The workcell software library consists of four major classes of routines. These are: vision and parts recognition, workcell device control, utilities, and "task-level" functions. All are written in V+. The "task-level" functions are of greatest interest to our work. These correspond more or less to the operations found in the high-level assembly plan, which are pick and place, invert, and weld. Since our workcell does not have a welder, the weld operations become no-ops.

Pick and place tasks are implemented by two functions, one for the parallel-jaw gripper and the other for the vacuum gripper. The first has parameters for the pickup and placement locations and approach and depart heights. The vacuum gripper pick and place function has an additional parameter, the grasp offset (with respect to the part center). This parameter is required so that the gripper is not put on a hole.

The function implementing part and subassembly inversion has parameters for offsets in x, y, and θ. These are used in presenting the part to the inverter, so that the gripper does not collide with the inverter. The part is moved to the position specified by the offsets (which are relative to the center of the inverter jaws), and the inversion cycle is begun.

The software library consists of about 180 kilobytes of V+ code. About half is used for vision and recognition routines. No checking for success of the operations is performed.

2.4.5 Illustrator

The simplest graphical output, "flying parts", has already been mentioned.

The Archimedes 2 Illustrator is a facility for viewing animations of assembly sequences carried out by a simulated robotic workcell. It is based on CimStation®, a software product of SILMA, that is expressly designed for 3D graphical simulation of manufacturing systems. CimStation includes extensive tools for describing workcells, and can import data from Pro/ENGINEER to obtain part models. It is therefore well-suited for producing animations of robotic assemblies.

Illustrator control is implemented as a finite-state machine, with a state for each operation the workcell supports. Because it was designed to allow animation of arbitrary workcells performing arbitrary assemblies, it requires, as input, an assembly sequence *and* a workcell description. The latter can include

- geometrically and kinematically accurate robot models,
- models of grippers, fixtures, and ancillary equipment (such as inverters), and
- simulation control

3 System Performance

We have run the planners on a variety of examples; illustrations of three of the assemblies are shown in figures 3, 4, and 5. Table 1 summarizes planning times required by the two planners for various assemblies. Times were computed by running the planner, under a graphical user interface, on a Silicon Graphics Indigo2 200 MHz R4400 computer with Extreme graphics; times marked with an asterisk were run on an SGI Indigo2 100 MHz R4000 computer with Extreme graphics, without the graphical user interface. To calculate ACIS data size, the data for each distinct part is counted only once, regardless of the number of times that part appears in the assembly. Planning times are elapsed times necessary to load the pre-facetted data, identify all contacts in the assembly, and compute the plan.

The first figure shows a pattern wheel assembly, a main example handled by the original Archimedes system [13]. The pattern wheel has 13 parts, assembled unidirectionally, and is fastened together by laser welds. The pattern wheel was modeled with detailed welding specifications in the design module, and translated to ACIS format. The optimizing planner then determined an assembly plan for the pattern

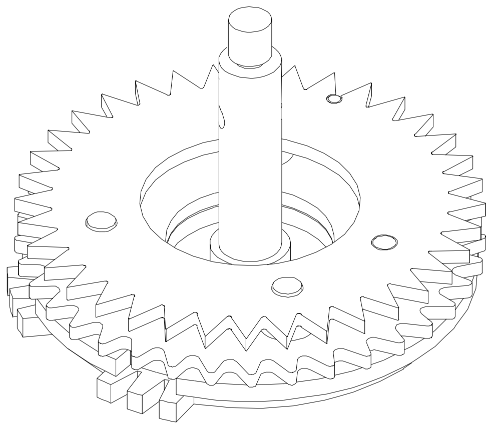

Figure 3: The pattern wheel assembly

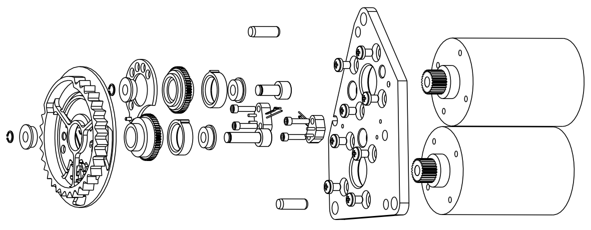

Figure 4: The discriminator

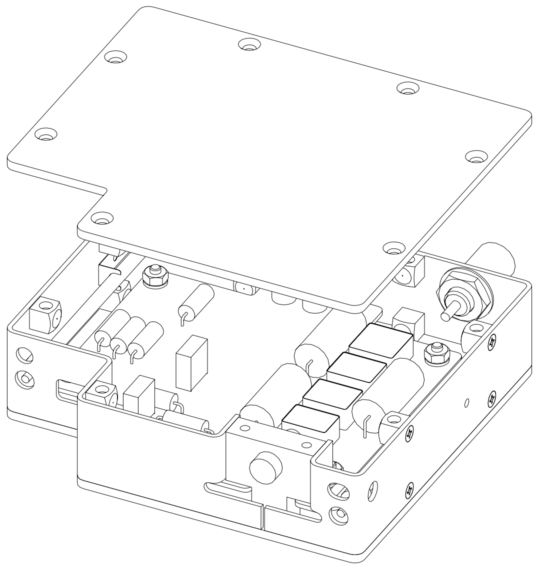

Figure 5: The Rockwell assembly

	N	MB	GE	OP
switchtube	5	0.2	7 sec	8 sec
pattern wheel	13	0.5	14	36
sprytron	14	1.1	17	33
neutron generator	35	1.5	51	—
Rockwell	78	3.2	167	—
fuel pump	36	3.6	175	—
discriminator	42	4.1	77	96*
locker	109	4.5	45*	—
seeker	70	6.1	120*	—
accelerometer	79	16	167	—
door lock	27	41	16 min*	—

Table 1: Planning times for various assemblies. "N" is the number of parts in the assembly, "MB" is the megabytes of ACIS data, "GE" is the geometric engine planning time, and "OP" is the optimizing planner planning time. An "*" indicates timing under different conditions; see text.

wheel. Since assembly reorientations and robot gripper changes are by far the slowest operations in the workcell, the optimality criterion given to the planner minimizes the number of these operations in the plan (as was discussed in section 2.4.2). The resulting plan was illustrated in a simulated workcell, and the plan was automatically translated to V+ code, using the library described above. The resulting robot program was executed in the workcell to assemble the pattern wheel. Laser welds were not performed automatically; instead, pre-welded subassemblies were substituted in the workcell when the program specified a laser weld.

To date, the pattern wheel is the only assembly which has exercised all modules of Archimedes 2.

Interested readers can retrieve color pictures of the examples, animations of sequences, illustrator output, and video of the robotic workcell assembling the pattern wheel on the World-Wide Web (`http://www.sandia.gov/2121/archimedes/archimedes.html`).

4 Current and Planned Work

The assembly planning problem can be addressed at many levels of detail. Fully specified assembly plans must satisfy many constraints and specify many details, ranging from part accessibility, to stability and fixturing, to fine motion plans, to the people or technology targeted to assemble the product. A more aggressive definition would also include factory layout and scheduling, inventory control, and many other factors. In industry, a detail in any of these areas might drive a choice of assembly plan [11]. Hence, to be useful, an assembly planning system must somehow enable generation of the "right" plan according to all these constraints.

However, most of these are active areas of research in themselves. Even if techniques existed to evaluate them, constructing a system that integrated all the constraints on assembly plans would be a huge effort, and it is not at all clear how it should be structured.

We are currently working on encoding and using constraints expressing tool accessibility, workcell capabilities, and grasping. A large catalog of relevant constraints can be found in [5].

Making the planner more interactive is also a major goal. Users could specify constraints to honor, part groupings, and subassemblies more easily in an interactive system than is currently possible. "Obviously" fruitless areas of the search space could be eliminated by the user. We have already implemented a graphical user interface under which all the facilities of Archimedes 2 are available. We plan to use this interface as the mechanism of interaction with the user.

5 Conclusion

We have described Archimedes 2 and its capabilities, as determined by experimentation on models of real mechanical assemblies. Based on its performance, we believe that it represents the state of the art in implemented assembly planning systems, at least in terms of the complexity of assemblies for which plans have been successfully computed. Two aspects of the system, in particular, have allowed us to achieve these results: the fast collision detection algorithm; and the declaration of part matings that are threaded or require part deformation, in place of sophisticated feature-recognition software. This latter point suggests that human input to the assembly planning process is critical, and that future research should address finding the proper balance between human and machine effort.

References

[1] A. Bourjault. *Contribution à une approche méthodologique de l'assemblage automatisé: élaboration automatique des séquences opératoires.* PhD thesis, Faculté des Sciences et des Techniques de l'Université de Franche-Comté, 1984.

[2] L. S. Homem de Mello and S. Lee, editors. *Computer-Aided Mechanical Assembly Planning.* Kluwer Academic Publishers, 1991.

[3] L. S. Homem de Mello and A. C. Sanderson. A correct and complete algorithm for the generation of mechanical assembly sequences. *IEEE Trans. on Robotics and Automation*, 7(2):228–240, 1991.

[4] ISO. *ISO 10303: Product Data Representation and Exchange*, 1994.

[5] R. E. Jones and R. H. Wilson. A survey of constraints in automated assembly planning. In *Proc. of the 1996 IEEE Int. Conf. on Robotics and Automation*, 1996.

[6] S. G. Kaufman, R. H. Wilson, R. E. Jones, T. L. Calton, and A. L. Ames. LDRD final report: Automated planning and programming of assembly of fully 3D mechanisms. Technical report, Sandia National Laboratories, 1995. In review.

[7] H. Ko and K. Lee. Automatic assembling procedure generation from mating conditions. *Computer Aided Design*, 19(1):3–10, 1987.

[8] L. I. Lieberman and M. A. Wesley. AUTOPASS: An automatic programming system for computer controlled mechanical assembly. *IBM J. of Research and Development*, 21(4):321–333, 1977.

[9] T.-H. Liu and G. W. Fischer. Developing feature-based manufacturing applications using PDES/STEP. *Concurrent Engineering: Research and Applications*, 1(1):39–50, 1993.

[10] T. Lozano-Pérez and P. H. Winston. LAMA: A language for automatic mechanical assembly. In *Proc. of the Intl. Joint Conf. on Artificial Intelligence*, 1977.

[11] J. L. Nevins, D. E. Whitney, T. L. De Fazio, A. C. Edsall, R. E. Gustavson, R. W. Metzinger, and W. A. Dvorak. *Concurrent Design of Products and Processes.* McGraw-Hill, 1989.

[12] N. J. Nilsson. *Principles of Artificial Intelligence.* Tioga Publishing, 1980.

[13] D. R. Strip and A. A. Maciejewski. Archimedes: An experiment in automating mechanical assembly. In *Proc. of the 11th ASME Intl. Conf. on Assembly Automation*, 1990.

[14] R. H. Wilson. *On Geometric Assembly Planning.* PhD thesis, Stanford Univ., March 1992. Stanford Technical Report STAN-CS-92-1416.

[15] R. H. Wilson. A framework for geometric reasoning about tools in assembly. In *Proc. of the 1996 IEEE Int. Conf. on Robotics and Automation*, 1996.

[16] R. H. Wilson and J.-C. Latombe. Geometric reasoning about mechanical assembly. *Artificial Intelligence*, 71(2):371–396, 1994.